Exploring Correlation-Driven Feature Selection for Mushroom Classification

Data Set: Mushroom

Kranti Vasant Adsul
kadsul@usc.edu

Sudarshana Sudheendra Rao
ssudheen@usc.edu

May 2, 2023

1 Abstract

This project aims to analyze the effect of different feature engineering and dimensionality adjustment techniques (performed on the mushroom dataset) on the performance of various Machine Learning models. The mushroom dataset is a collection of 173 different species of 61,069 mushrooms. The dataset contains 15 features including a class labels column which indicates whether the mushroom is poisonous or edible. The dataset has to be pre-processed as the features are made up of both numerical and categorical values. The pre-processed dataset was used to conduct 2 experiments. Experiment 1 involved using correlation among the features to engineer new features followed by using Principal Component Analysis to extract features having the highest variances. While Experiment 2 involved using the Univariate Feature Selection technique to capture important features in the pre-processed dataset. Five Machine Learning models were implemented- Logistic Regression, Support Vector Machine, Gaussian Naive Bayes Classifier, Random Forest Classifier, and Multi-layer Perceptron and were used to analyze the effect of feature engineering using correlation coefficients on the model performance. Cross-validation was also performed on the models, and the optimal model was selected based on the best performance on the test dataset. It was found that Random Forest Classifier was the best model with a testing accuracy of 94.30% when used on the dataset generated using pre-processing, feature engineering, and PCA (Experiment 1); Support Vector Machine was the best model with a testing accuracy of 73.10% when used on the dataset generated using pre-processing and UFS (Experiment 2). It was concluded that feature engineering using correlation coefficients helps to improve the performance of the models, but requires careful selection of the components by controlling the hyperparameters of feature reduction techniques, ignorance of which can easily lead to the overfitting of models.

2 Introduction

The mushroom dataset is a well-known dataset in the field of Machine Learning as it is extensively used for classification. The mushrooms are classified as poisonous or edible based on the 15 features present in the dataset. The features are a mixture of numerical and categorical values, therefore many pre-processing, feature engineering, and dimensionality adjustment techniques can be used on this dataset. This dataset is popularly used to evaluate the performance of various Machine Learning models and recently, Neural Networks are being used on this dataset. The effect of different feature engineering and dimensionality reduction techniques on the performance of the five implemented Machine Learning models was analyzed in this project.

2.1 Problem Assessment and Goals

The mushroom dataset is comprised of 12 categorical features and three numerical features, therefore the dataset has to be pre-processed. The numerical features have to be standardized, and the categorical features have to be one-hot encoded. As a result of one-hot encoding, several new features would be added

and thereby increasing the total number of features. Hence, the dimensionality of the features needs to be adjusted either by identifying the important features and dropping the redundant ones or by combining a few features through feature engineering. Later, different Machine Learning models have to be trained, and the best model has to be selected based on the performances.

The goals of this project are:

1. To implement a trivial and a baseline system that will provide a benchmark performance.
2. To perform cross-validation on the training dataset and to train the different models.
3. To compare the models' performances with each other and with the benchmark.
4. To select the best model and to test it on the testing dataset.

3 Approach and Implementation

The methodology followed in this project is described in this section. The project can be broadly classified into two experiments and subsection 3.4.2 explains this in detail. Subsections 3.1 through 3.4 explains the analysis and pre-processing carried out on the dataset, and subsection 3.5 explains the different models implemented.

3.1 Dataset Description and Usage

Datasets (training and testing) consist of numerical and categorical data. The CSV files consist of 15 features of 173 different species of mushrooms (61069 mushrooms). The 15 features are stated in the table 1.

Index	Features
1	cap-diameter
2	cap-shape
3	cap-surface
4	cap-color
5	does-bruise-or-bleed
6	gill-attachment
7	gill-spacing
8	gill-color
9	stem-height
10	stem-width
11	stem-color
12	has-ring
13	ring-type
14	habitat
15	season

Table 1: List of Features

The last column is the class label 'p' and 'e'. 'p' stands for poisonous class, and 'e' stands for edible class. The following features have categorical attributes:

- cap shape has b- bell, c- conical, x- convex, f- flat, k- knobbed, and s- sunken.
- cap surface has f- fibrous, g- grooves, y- scaly, and s- smooth.
- does-bruise-or-bleed has t- bruises-or-bleed, and f- no.
- gill attachment has a- attached, d- descending, f- free, and n- notched.
- gill spacing has c- close, w- crowded, and d- distant.

- gill color has k- black, n- brown, b- buff, h- chocolate, g- gray, r- green, o- orange, p- pink, u- purple, e- red, w- white, and y- yellow.
- stem color has n- brown, b- buff, c- cinnamon, g- gray, o- orange, p- pink, e- red, w- white, and y- yellow.
- ring type has c- cobwebby, e- evanescent, f- flaring, l- large, n- none, p- pendant, s- sheathing, z-zone.
- has-ring has f- false, and t- true.
- habitat has g- grasses, l- leaves, m- meadows, p- paths, u- urban, w- waste, and d- woods.
- season has s- spring, u- summer, a- autumn, and w- winter.
- cap color has brown- n, buff- b, gray- g, green- r, pink- p, purple- u, red- e, white- w, yellow- y, blue- l, orange- o, and black- k. [1]

The training dataset CSV file containing 42,746 data points (excluding the headers) under each feature column was split (using Sklearn's '*train_test_split*' method) into two- a new (smaller) training dataset containing 34,197 data points in each feature and a validation dataset containing 8549 data points in each feature. Cross-validation was employed to evaluate the model's performance in each fold with different training and validation datasets subsets. Sklearn's '*cross_val_score*' function was used in experiment-1, whereas the 'KFold' function was used in experiment-2 (experiments 1 & 2 are explained in the later part of the report). The number of folds chosen was five (k), and the new training dataset was shuffled and split into random five subsets in every fold. The '*cross_val_score*' function computes the accuracy of the model's prediction in each fold and returns the average of the accuracies; on the other hand, the model's accuracy was computed in every fold while using the 'KFold' function [2]. Cross-validation was performed on all the implemented except the trivial system. The cross-validation was used in five epochs for the Multi-layer Perceptron.

The testing dataset contains 18,321 data points under each feature column. The testing dataset was used twice, once on the Support Vector Machine in experiment-1 and once on the Random Forest Classifier in experiment-2. In experiment-2, the validation accuracies of all the models were plotted on a single line graph for comparison. The Random Forest Classifier's performance was the best, so the testing dataset was used on it.

3.2 Exploratory Data Analysis

The dataset was checked for any missing/void/null values using the Pandas "isnull().sum()" method and no such values were observed. Figure 1 illustrates the first five rows of the dataset, and figure 2 depicts the datatypes of each feature.

	cap-diameter	cap-shape	cap-surface	cap-color	does-bruise-or-bleed	gill-attachment	gill-spacing	gill-color	stem-height	stem-width	stem-color	has-ring	ring-type	habitat	season	class
0	4.98	c	i	y	f	a	c	n	6.04	6.21	w	f	f	d	a	p
1	2.84	x	y	y	f	a	c	w	5.66	3.55	y	t	r	h	u	p
2	11.44	x	y	y	f	a	c	w	7.03	25.29	n	t	e	d	w	e
3	8.77	s	t	r	t	d	c	g	4.44	13.61	r	f	f	d	a	p
4	7.55	x	d	n	t	p	c	y	8.41	18.44	y	f	f	d	a	e

Figure 1: Initial Rows of the Mushroom Dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42748 entries, 0 to 42747
Data columns (total 16 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   cap-diameter          42748 non-null  float64
 1   cap-shape             42748 non-null  object
 2   cap-surface           42748 non-null  object
 3   cap-color             42748 non-null  object
 4   does-bruise-or-bleed  42748 non-null  object
 5   gill-attachment       42748 non-null  object
 6   gill-spacing          42748 non-null  object
 7   gill-color            42748 non-null  object
 8   stem-height           42748 non-null  float64
 9   stem-width            42748 non-null  float64
 10  stem-color            42748 non-null  object
 11  has-ring              42748 non-null  object
 12  ring-type             42748 non-null  object
 13  habitat               42748 non-null  object
 14  season                42748 non-null  object
 15  class                 42748 non-null  object
dtypes: float64(3), object(13)
memory usage: 5.2+ MB
```

Figure 2: Datatypes of the Class Features

The number of poisonous mushrooms was 23595, and the number of edible mushrooms was 19153. Therefore, the probability of picking a poisonous mushroom is 0.5393, and that of picking an edible mushroom is 0.4607. Later as a part of the pre-processing step (described below), two classes were label-encoded, and the distribution of the classes was visualized by a bar graph. Here 'e' (edible mushrooms) is assigned the value '0', and class 'p' (poisonous mushrooms) is assigned the value 1.

Figure 3: Distribution of the Two Mushroom Classes

Figure 3 concludes that the dataset is almost balanced. Furthermore, the number of occurrences of each categorical data was recorded in table 2.

Features	Categorical Data	Number of occurrence
cap-shape	c	1288
	x	18808
	s	5079
	o	2417
	f	9289
	b	4043
	p	1824
cap-surface	i	1599
	y	4460
	t	15569
	d	3125
	g	3280
	h	3482
	k	1602
	s	5341
	e	1789
	w	1494
	l	1007
cap-color	y	6026
	r	1227
	n	16960
	w	5402
	g	3072
	k	907
	o	2581
	l	580
	e	2781
	u	1181
	b	870
	p	1161
does-bruise-bleed	f	35279
	t	7469
gill-attachment	a	15844
	d	7133
	p	4185
	x	5174
	e	3998
	f	2501
	s	3913
gill-spacing	c	34791
	d	5456
	f	2501
ring-type	f	35581
	r	1013
	e	1703
	z	1418
	l	998
	p	907
	g	888
	m	240
has-ring	f	32085
	t	10663

Features	Categorical Data	Number of occurrence
gill-color	n	6742
	w	12893
	g	2860
	y	6743
	p	4178
	k	1697
	f	2501
	b	662
	e	756
	o	2041
	r	962
	u	713
stem-color	w	15996
	y	5547
	n	12678
	r	383
	o	1512
	g	1841
	l	150
	e	1431
	u	1031
	f	745
	k	599
	p	710
	b	125
habitat	d	30945
	h	1444
	g	5486
	l	2244
	m	2041
	w	259
	p	245
	u	84
season	a	21053
	u	16065
	w	3669
	s	1961

Table 2: Description of the Dataset

3.3 Preprocessing

The dataset comprises the following numerical and categorical values distribution as described in table 3.

Type of data	Number of features
Numerical data	3
Categorical data	13

Table 3: Number of Features in the Two Types of Data

Firstly, the non-numerical data points were converted to numerical data points by one-hot encoding. One-hot encoding is a process of converting categorical variables into a binary vector format, where each category is represented by a unique binary value. In this format, each feature is represented by a binary vector where all of the elements are zero except for the one corresponding to the feature value. This was achieved by using the Pandas inbuilt function `get_dummies()` which creates new binary columns for each categorical value in the original columns. The original dataset was transformed, and table 4 explains the distribution of transformed features:

Feature	Number of types
cap-shape	7
cap-surface	11
cap-color	12
does-bruise-bleed	2
gill-attachment	7
gill-spacing	3
gill-color	12
stem-color	13
has-ring	2
ring-type	8
habitat	8
season	4
class	2
Total	**91**

Table 4: Count of Unique Values in each Feature

`get_dummies()` function added 91 columns corresponding to the different types of features as stated above to the original data set to represent the one-hot encoded format. An additional benefit of one-hot encoding is to remove outliers in the data; figures 4 and 5 are the bar graphs representing the distribution of the categorical data before and after one-hot encoding:

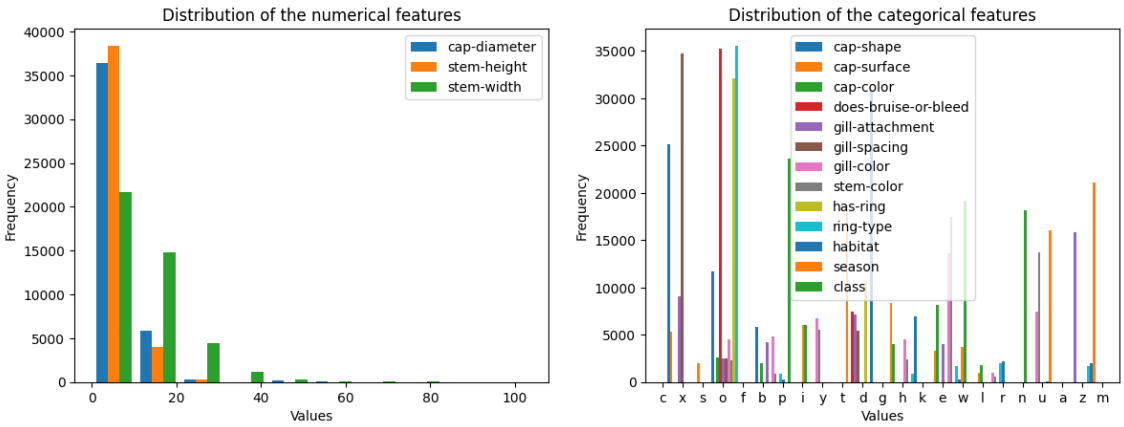

Figure 4: Histograms of the Original Features

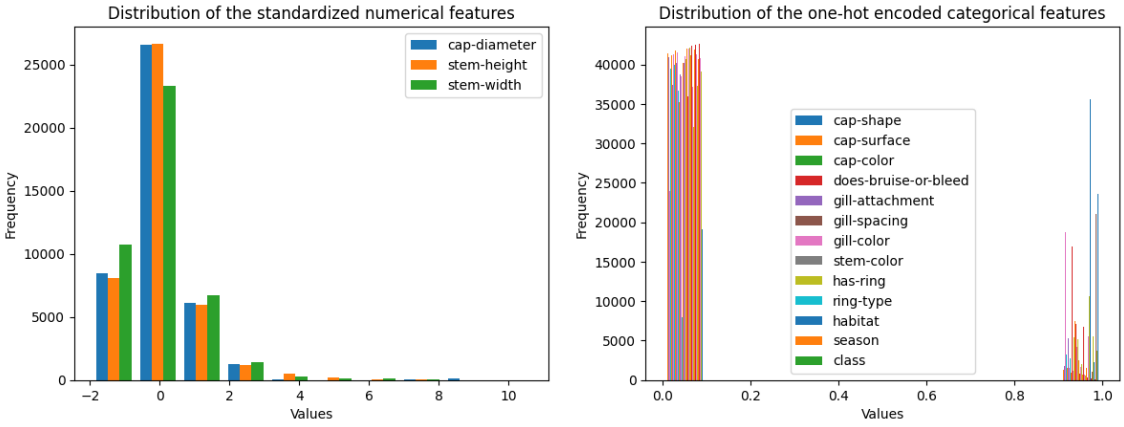

Figure 5: Histograms of the Transformed Features

Further, the numerical data ('cap-diameter', 'stem-width', and 'stem-height') was normalized by using StandardScalar class from scikit-learn. Standard scaling is a pre-processing step used to normalize the input features. It helps to transform the features to have a mean of zero and a standard deviation of one. Standard scaling was used to ensure that the input features are on the same scale, which helps to improve the performance of the Machine Learning models. The below formulas describe how the normalization works for a vector/feature 'X':

The mean of X is calculated as:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

The standard deviation of X is calculated as follows:

$$s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{X})^2}$$

StandardScalar standardizes the data using the formula:

$$X_{\text{standardized}} = \frac{X - \bar{X}}{s}$$

where $X_{\text{standardized}}$ is the standardized version of X, X is the original vector of data, $\bar{X}$ is the mean of X, and s is the standard deviation of X. [3]

Additionally, Support Vector Machines (used in this project) are sensitive to the scale of the input features, and standard scaling helps to ensure that the model works correctly. The effect of standard scaling is illustrated in figure 6, which shows the standard deviation and mean of the three numerical features before and after applying the standard scaling to the data.

Original Data Statistics:

	Statistic	Cap Diameter	Stem Height	Stem Width
0	Mean	6.714149	6.583224	12.117692
1	Standard Deviation	5.220008	3.368333	10.004874
2	Minimum	0.380000	0.000000	0.000000
3	25th Percentile	3.480000	4.640000	5.180000
4	Median	5.865000	5.960000	10.200000
5	75th Percentile	8.530000	7.750000	16.540000
6	Maximum	62.340000	33.920000	103.910000

Scaled Data Statistics:

	Statistic	Cap Diameter	Stem Height	Stem Width
0	Mean	-2.940579e-16	1.759299e-17	1.877183e-16
1	Standard Deviation	1.000012e+00	1.000012e+00	1.000012e+00
2	Minimum	-1.213451e+00	-1.954469e+00	-1.211193e+00
3	25th Percentile	-6.195751e-01	-5.769166e-01	-6.934393e-01
4	Median	-1.626738e-01	-1.850268e-01	-1.916780e-01
5	75th Percentile	3.478678e-01	3.463996e-01	4.420205e-01
6	Maximum	1.065640e+01	8.115913e+00	9.174866e+00

Figure 6: Comparison of Data Before and After Normalization

Finally, pre-processing resulted in 94 columns- 91 one-hot encoded columns and three standardized numerical data columns. The two columns representing the one-hot encoding of the two classes- 'poisonous' and 'edible' mushrooms are combined into one column representing the true labels. These steps are followed to get the pre-processed testing dataset in the models described below.

3.4 Feature Engineering and Dimensionality Adjustment

In this section, methods used to develop new relevant statistical features were analyzed. Pearson's Correlation Coefficient was used to analyze the inter-dependency and correlation between the 15 features. Dimensionality adjustment of the dataset was achieved by employing Principal Component Analysis and Univariate Feature Selection.

3.4.1 Pearson Correlation Coefficient

The Pearson correlation coefficient is a statistical test that measures the strength and direction of the linear relationship between two variables based on covariance. The correlation coefficient ranges from +1 to -1, indicating a perfect correlation if the value is near these extremes and no correlation if the value is 0. It is symmetric and independent of the unit of measurement. For feature selection, both positively (when one increases, then the other also increases) and negatively (when one increases, the other decreases) correlated features should be considered [4]. The numerical features having the top ten positive and negative coefficient values feature where at least one of the features is- stem height/stem width/cap diameter were identified for feature engineering purposes. Figure 7 depicts the outcome of the analysis.

Top 10 positive correlations:

	Feature-1	Feature-2	Correlation coefficient
3	cap-diameter	stem-width	0.695804
4	stem-width	cap-diameter	0.695804
5	stem-height	stem-width	0.436069
6	stem-width	stem-height	0.436069
7	stem-height	cap-diameter	0.423171
8	cap-diameter	stem-height	0.423171
9	ring-type_m	stem-height	0.406376
10	gill-attachment_p	stem-width	0.400621
11	gill-attachment_p	cap-diameter	0.353896
12	ring-type_f	stem-height	0.332793

Top 10 negative correlations:

	Feature-1	Feature-2	Correlation coefficient
275	gill-color_o	stem-width	0.000822
274	cap-color_e	cap-diameter	0.000841
273	stem-color_u	stem-height	0.000941
272	ring-type_l	stem-width	0.001029
271	habitat_u	stem-width	0.001503
270	cap-surface_w	stem-width	0.001601
269	gill-spacing_c	stem-width	0.001994
268	habitat_l	stem-width	0.002066
267	has-ring_t	stem-width	0.002306
266	has-ring_f	stem-width	0.002306

Figure 7: Top Ten Correlation Coefficients of the Class Features

Using the values shown in figure 7, it was decided to add the minimum, maximum, and average values of the combinations of features as described in table 5.

Index	Feature combination	New Features Added
1	'ring-type_m', 'stem width'	stem_width_ring_type_m_min stem_width_ring_type_m_max stem_width_ring_type_m_mean
2	'gill-attachment_p', 'stem width'	stem_width_gill_attachment_p_min stem_width_gill_attachment_p_max stem_width_gill_attachment_p_mean
3	'ring-type_f', 'stem height'	stem_height_ring_type_f_min stem_height_ring_type_f_max stem_height_ring_type_f_mean
4	'gill-color_o', 'stem width'	stem_width_gill_color_o_min stem_width_gill_color_o_max stem_width_gill_color_o_mean
5	'cap-color_e', 'cap diameter'	cap_diameter_cap_color_e_min cap_diameter_cap_color_e_max cap_diameter_cap_color_e_mean
6	'cap-shape_p', 'cap diameter'	cap_diameter_p_min cap_diameter_p_max cap_diameter_p_mean

Table 5: Additional Features due to Feature Engineering

These new features were added as new columns to the original dataset giving a total of 111 columns- three columns (normalized numerical data), 89 one-hot encoded columns, one class label column, and 18

columns (new combinations added above), where the total number of actual features excluding the class label column is equal to 110. Training a model with such a huge number of features will increase time and space complexity. Also, it is essential to ensure that the number of data points (N) exceeds 3-10 times the number of independent parameters the model learns (d.o.f). Failure to meet this condition may result in overfitting (the model fails to generalize as it learns the patterns and the noise of the training dataset so well that, when evaluated on an unseen test dataset will result in low accuracy). Therefore, dimensionality reduction is essential to identify important features, remove redundant ones, and enhance the model's performance.

3.4.2 Feature Dimensionality Reduction

In this project, two experiments were conducted to analyze the effect of feature engineering on the performances of the implemented models and, ultimately, to compare the model's performance. The first experiment involves the usage of Principal Component Analysis (PCA) on the pre-processed and feature-engineered dataset. The second experiment involves the usage of Univariate Feature Selection (UFS) on the dataset obtained after pre-processing. Both the feature reduction techniques- PCA and UFS, were implemented to select the best features from the feature-engineered dataset.

3.4.3 Univariate Feature Selection

UFS is a technique in Machine Learning that involves selecting the most important features in a dataset based on their relationship with the target variable. The idea behind UFS is to evaluate each feature independently, without considering the relationship between features, and then select the subset of features that are most strongly correlated with the target variable. SelectKBest class from the sci-kit-learn library was used to perform UFS with the chi-squared test as the scoring function. The SelectKBest class selects the k top features based on their scores from the chi-squared test. The chi-squared test is used to measure the dependence between each feature and the target variable in a classification problem. It evaluates whether the occurrence of a particular feature value is significantly different for different values of the target variable.

Moreover, the sorted chi-scores of the original features were retrieved using `ufs.scores_` and were plotted against 20 features. This gave a rough idea of the number of features required for a model to perform well. It is evident from figure 8 that the chi-score relatively decreases after seven features, so the value of `k` was selected as seven while using UFS. [5]

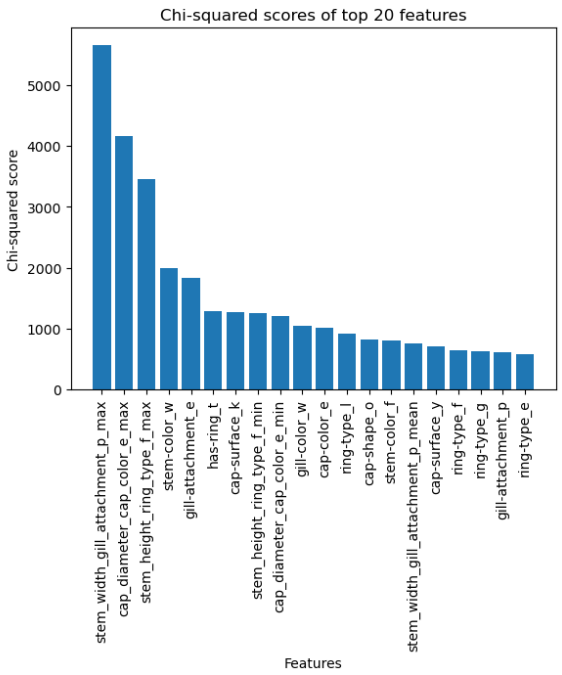

Figure 8: Chi Scores versus Number of Features

3.4.4 Principal Component Analysis

PCA is a technique used in Machine Learning and data analysis to reduce the dimensionality of a large dataset while retaining the important features. This is achieved by transforming the original features of the dataset into a new set of features (principal components), which are linear combinations of the original features. The principal components are ordered by the amount of variance in the original dataset- the first principal component retains the highest portion of the dataset's variance, and each subsequent components retain the maximum amount of leftover variance in the decreasing order of importance. This allows for representing the data in a lower-dimensional space while retaining most of the important information. The sorted variances of each principal component (transformed features) were retrieved using `pca.explained_variance_` and plotted against the total number of features to obtain the number of features required for a model to perform well.

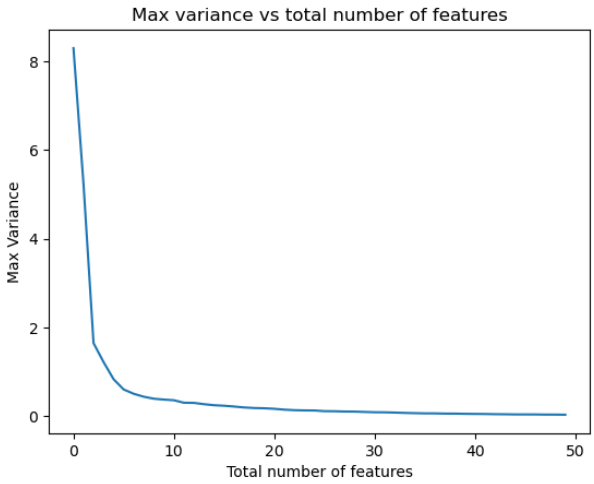

Figure 9: Graph of Maximum Variance contributed by each Feature

Figure 9 shows the sharp decrease in the "maximum variance" (contributed by the features) after 10 features, so the value of `n_components` was selected as 10 while using PCA. [6]

3.5 Training, Classification and Model Selection

A total of five models, namely, Logistic Regression, Support Vector Machine, Random Forest Classifier, Multi-layer Perceptron, and Gaussian Naive Bayes Classifier, were implemented in this project. All five models were used in experiments 1 & 2. The trivial and the baseline systems were used as a basis for comparing the performances of each model. The Random Forest Classifier and the Gaussian Naive Bayes Classifier are the two **non EE-559** models implemented in this project. The validation accuracies of the five models used in experiment-2 were plotted on a single line graph. The best model was selected based on the highest accuracy achieved; Random Forest Classifier was the best-performing model. On the contrary, the best-performing model in experiment-1 was Support Vector Machine.

3.5.1 Baseline System

Nearest Means Classifier (NMC) was used as the baseline system in this project. NMC is a simple classification algorithm that calculates the mean feature vector for each class in the training set and then classifies new data points based on their proximity to the class means. NMC calculates the mean vectors for both the classes ('e' and 'p') by averaging the feature vectors of all the training samples in the class, and data points are classified by computing its Euclidean distance to each class mean. The Euclidean distance used by NMC is given by the formula:

$$d(x, C_{center}) = \sqrt{\sum_{j=1}^{n} (x_j - C_{center})^2}$$

where x_j is the data point, 'n' is the number of data points, and 'd' is the Euclidean distance. If 'd' for a particular 'x' is close to the 'C_{center}' of class 'e', then the NMC will classify 'x' as class 'e'; else, 'x' will be classified as class 'p'. Cross-validation was performed on this model, and the recorded training and validation accuracies were 63.08% and 62.65%, respectively. This model was tested using the new testing data frame, and the model performance was analyzed using 'accuracy_score' and 'F1 score'. The model was found to have an accuracy of 62.85% on the test set and an F1-score of 0.67. [7]

3.5.2 Trivial System

The trivial system used in this project outputs the two class labels- 'p' & 'e' at random (on the testing dataset) using the respective probabilities:

$$S_0 = \frac{N_0}{N} \qquad\qquad S_1 = \frac{N_1}{N}$$

where N is the number of rows present in the training dataset, and N_i is the number of training data points belonging to the respective class label S_i. This was achieved by using Numpy's "random.choice()" method, which randomly assigns the class labels to the testing data points using S_0 & S_1. The test accuracy of this trivial system was 51%, and the F1 score for class p was 0.55 and for class e 0.44. It was concluded that the trivial system was not accurate in classification as the accuracy and the F1 scores were low. Figure 10 depicts the confusion matrix (plotted using Seaborn's heatmap function) of the trivial system. The number of true positives was relatively low, and the number of misclassified data points was high, adding to the fact that the trivial system does not accurately classify the data.

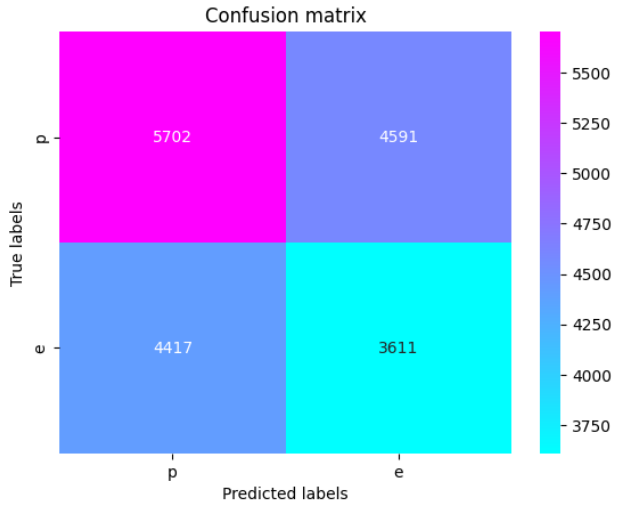

Figure 10: Confusion Matrix of the Trivial System

3.5.3 Logistic Regression

Logistic Regression is an algorithm used for binary classification problems, where the task is to predict the class of a data point as one of two possible classes. It's a type of generalized linear model that uses a logistic function to model the probability of the binary output. The logistic function, also called the sigmoid function, maps any input value to a value between 0 and 1. The Logistic Regression algorithm uses this sigmoid function to transform the linear regression output into a probability.

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-z}}$$

where y is the binary output variable (0 or 1), x is the vector of input features, and z is the linear combination of input features and corresponding weights, given by:

$$z = w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

where w_i are the weights of the model and x_i are the input features.

Logistic Regression aims to estimate the optimal values of the weights w_i that minimize the difference between the predicted probability and the actual class labels in the training data. This is usually done by maximizing the log-likelihood function or minimizing the binary cross-entropy loss function. During the prediction stage, the logistic regression algorithm calculates the linear combination of the input features and the learned weights and then applies the sigmoid function to the result to obtain the predicted probability. If the predicted probability is greater than a certain threshold value (0.5 in sci-kit-learn), the output is predicted as class 1; otherwise, it's predicted as class 0. Logistic Regression was implemented in this project using the inbuilt LogisticRegression() class from the sci-kit-learn library. [8]

3.5.4 Support Vector Machine

SVM algorithm is used for classification and regression analysis. SVM works by finding the optimal hyperplane that separates the data points into different classes. In SVM, the data points are mapped to a high-dimensional space, and the hyperplane is found as the one that maximizes the distance between the closest points from each class. These closest points are called support vectors; the margin between them is called the maximal margin. The idea is that the larger the margin, the better the separation between the classes. There are different types of SVM, such as linear SVM, polynomial SVM, and radial basis function SVM. The inbuilt SVC class from sci-kit-learn with RBF kernel was used for this project as the dataset is not linearly separable (because categorical values and the data points are not continuous). This SVM model makes use of a Gaussian similarity function, which computes the similarity between a data point from the validation set and the data point from the training set by using the formula:

$$K(x_1, x_2) = \exp\left(\frac{|x_1 - x_2|^2}{2\sigma^2}\right)$$

SVM calculates the Euclidean distance of the data points, say x_1 and x_2, which are normalized by σ (bandwidth parameter). Further, the Euclidean distance is transformed to a similarity measure (ranging between 0 to 1) by the $\exp(\cdot)$ function. Different values of 'C' and 'gamma' (hyperparameters) were experimented with, ranging from 0.01 to 1 and 1 to 5, respectively. The optimal results were obtained with C = 1 and gamma = 3. [9]

3.5.5 Multi-layer Perceptron

MLP is a type of Artificial Neural Network where perceptrons (neurons) in each layer are connected to all perceptrons of the subsequent layer. The first layer of the MLP is the input layer, followed by a number of linear layers, and the final layer is the output layer with the activation function. Usually, for binary classification, sigmoid is used as the activation function, as it outputs a value between 0 and 1. The MLP computes the weights and biases for every perceptron in each layer. The weight propagation to the subsequent layers is determined by the loss function, type of regularization, and the choice of optimizer. The MLP used in this project was a feed-forward model and was implemented using TensorFlow's Keras framework. Binary cross entropy was the loss function used with adam optimizer, and the batch size of the validation set in each of the five epochs was 32. The input layer consists of ten neurons, two linear layers with ReLU activation function consisting of 64 and 32 neurons, respectively, and one output neuron with sigmoid activation function. The ReLU activation function outputs according to the equation:

$$f(x) = \max(0, x)$$

where x is the input, and $f(x)$ is the output. The above equation returns the maximum value between 0 and x. The output of the sigmoid activation follows the equation:

$$f(x) = \frac{1}{1 + e^{-x}}$$

The binary cross entropy loss function is reduced using the formula:

$$L(y, \hat{y}) = -y * \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

where y is the true label, and $\hat{y}$ is the predicted probability of class-1. [10]

3.5.6 Random Forest Classifier

Random Forest Classifier is an ensemble learning method used for both classification and regression tasks. It is an extension of decision tree algorithms and consists of a large number of individual decision trees. Each decision tree is built on a random subset of the training data and a random subset of the input features. During the training process, the algorithm constructs multiple decision trees, each of which makes a prediction. Then, the predictions of all decision trees are combined to make a final prediction. This process of combining predictions is called "bagging" or "bootstrap aggregating." In the case of classification tasks, the final prediction is the mode of the predictions made by individual decision trees. The Random Forest Classifier was implemented using the RandomForestClassifier class from sci-kit-learn. The following steps are the algorithm followed by Random Forest Classifier:

1. k random points are taken from the training dataset.
2. Individual decision trees are constructed from each k-selected random point.
3. Each decision tree will generate an output.
4. Final class assigned based on majority voting [11]

Values of the hyperparameters found to give optimal results were: 'n_estimators = 100' and 'min_samples_split = 2'.

3.5.7 Gaussian Naive Bayes Classifier

Gaussian Naive Bayes model is a probabilistic (Bayes theorem) algorithm used for classification. Below are the steps followed by this algorithm:

1. Prior probability of the binary class is computed based on the frequency of the two classes.
2. Mean and standard deviation of all input features are calculated.
3. The probability density function of all input features is calculated using the mean and standard deviation values computed in step 2.
4. Posterior probability of the particular subset of input features (cross-validation) is calculated using Bayes' theorem.
5. The class with the highest posterior probability is predicted as the output.

The posterior probabilities are calculated by the equation:

$$P(y|x_1, x_2, ..., x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i|y)}{P(x_1, x_2, ..., x_n)}$$

where $P(y)$ is the prior probability of the class, $P(x_i|y)$ is the likelihood of the inputs features x_i given the class y, and $P(x_1, x_2, ..., x_n)$ is the marginal probability of the x_i. The output of the Naive Bayes Classifier is based on the equation:

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, P(y|x_1, x_2, ..., x_n)$$

where $\hat{y}$ is the output (predicted class). The model was implemented using sci-kit-learn's GaussianNB function. [12]

4 Analysis

The testing accuracy was used as the primary evaluation metric to assess the performance of each model and to choose the best model. The training and validation accuracy of each model was calculated in every fold of the cross-validation. The F1 scores of the models were calculated at the end of the cross-validation.

4.1 Evaluation Metrics

Validation accuracy involves the usage of the k-fold cross-validation technique with k = 5 in which the dataset is split into five sets where the first set is taken as the validation set, and the remaining sets are used for training, this split of the training and validation sets is updated during each fold. In each fold the model is fitted to the training set and evaluated on the validation set and an average validation accuracy is reported at the end. The following two evaluation metrics were used to evaluate the performance of the best model on the testing dataset:

1. F1- Score: F1-Score is the harmonic mean of the precision and recall.

$$\frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

2. Accuracy: The accuracy of the model is computed by the formula given below,

$$\frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{False Negative} + \text{True Negative} + \text{False Positive}}$$

Finally, a confusion matrix was used to visualize the classification behavior of the best model on the testing dataset. All of the above metrics were implemented using sci-kit-learn's metrics library.

4.2 Analysis of Outcomes

The results of the two experiments were analyzed in this subsection. Tables 6 and 7 state the results obtained when the principal components were equal to ten, whereas table 8 state the results obtained when the principal components were equal to 14. In tables 6 and 8, the accuracy values in the "Accuracy" column are the testing accuracies. The training and validation accuracies in table 7 is the highest accuracy obtained after cross-validation.

The effect of feature engineering on the model's test accuracies is evident in the tables below. The models start to overfit when the number of principal components exceeds ten (in table 8, testing accuracies of the models are worse compared to table 6's accuracies). One reason for overfitting upon increasing the number of features through PCA could be due to large dimensionality. As the number of features increases, the model has to deal with more noise and irrelevant information, which can lead to overfitting. Also, the model becomes too complex to generalize, which can lead to overfitting.

Additionally, the PCA transformation may not preserve all the important information in the original features, and some may be lost in the transformation. This loss of information can cause the model to overfit, especially if there is not enough data to properly capture the underlying patterns in the data.

Models	Experiment-1 dataset			Experiment-2 dataset		
	Confusion matrix	F1-score	Accuracy	Confusion matrix	F1-score	Accuracy
Baseline System	[[4488 3540][3265 7028]]	0.67	0.62	[[4488 3540][3265 7028]]	0.67	0.62
Trivial System	[[3668 4360][4572 5721]]	0.56	0.51	[[3668 4360][4572 5721]]	0.56	0.51
Random Forest	[[8916 536][537 8336]]	0.942	0.943	[[5800 3648][1683 7190]]	0.706	0.709
Logistic Regression	[[5896 3552][2317 6556]]	0.671	0.679	[[5773 3675][1759 7114]]	0.701	0.703
SVM	[[8887 561][710 8163]]	0.930	0.930	[[6776 2672][2243 6630]]	0.731	0.731
MLP	[[8250 1198][1216 7657]]	0.868	0.868	[[6581 2867][2076 6797]]	0.730	0.730
Naive Bayes	[[4527 4921][1660 7213]]	0.631	0.640	[[6123 3325][2587 6286]]	0.677	0.677

Table 6: Summary of Results Obtained with PCA (n_components = 10)

Models	**Validation accuracy**	**Training accuracy**
Logistic Regression	100%	100%
Random Forest Classifier	100%	100%
SVM	99.89%	100%
Naive Bayes Classifier	100%	100%
MLP	100%	100%

Table 7: Accuracies of the Models

Models	Experiment-1 dataset			Experiment-2 dataset		
	Confusion matrix	F1-score	Accuracy	Confusion matrix	F1-score	Accuracy
Baseline System	[[4488 3540][3265 7028]]	0.67	0.62	[[4488 3540][3265 7028]]	0.67	0.62
Trivial System	[[3668 4360][4572 5721]]	0.56	0.51	[[3668 4360][4572 5721]]	0.56	0.51
Random Forest	[[9448 0][0 8873]]	1.00	1.00	[[5800 3648][1683 7190]]	0.706	0.709
Logistic Regression	[[9448 0][0 8873]]	1.00	1.00	[[5773 3675][1759 7114]]	0.701	0.703
SVM	[[9448 0][10 8863]]	0.994	0.994	[[6776 2672][2243 6630]]	0.731	0.731
MLP	[[9448 0][0 8873]]	1.00	1.00	[[6581 2867][2076 6797]]	0.730	0.730
Naive Bayes	[[9448 0][0 8873]]	1.00	1.00	[[6123 3325][2587 6286]]	0.677	0.677

Table 8: Summary of Results Obtained with PCA (n_components = 14)

The accuracies of Logistic Regression and the baseline system were almost equal. On the contrary, the Naive Bayes Classifier performed poorly compared to the baseline system. The reason for this poor performance could be the dataset having a complex structure, and the assumption that the features are independent does not hold true; another reason could be the new features added by combining highly correlated features. The best-performing model was the Random Forest Classifier, as it doesn't assume anything about the data and works by using only a subset of features at the split of each decision tree. SVM is the next best-performing model because an RBF kernel transforms the input data into a higher-dimensional space where it is easier to find a separating hyperplane. MLP gives better test accuracy than the Baseline System, and the selection of the MLP's hyperparameters could be the reason for the improved accuracy.

To encapsulate, the best performing model was the Random Forest Classifier using the feature-engineered dataset, which gave a testing accuracy of 94%. This accuracy is an improvement of 32% over the Baseline System and 43% over the Trivial System. The Random Forest Classifier showed an F1-score of 0.942, which is 0.27 more than the Baseline System's F1-score and 0.38 more than that of the Trivial system.

5 Libraries

The list of libraries used to implement the models is enumerated in table 9.

Library	Model
Sci-Kit-Learn	Logistic Regression Support Vector Machine Gaussian Naive Bayes Classifier Random Forest Classifier
TensorFlow Keras	Multi-layer Perceptron

Table 9: Libraries used to Implement the Models

Furthermore, the Sci-kit-Learn library was used for the preprocessing techniques such as PCA, UFS, calculating the Pearson Correlation Coefficient, and standardizing the datasets. Also, the evaluation metrics such as F1 score, model accuracies, and confusion matrix were calculated using the Sci-kit-Learn library. One-hot encoding and label encoding were achieved by using the Pandas library.

The feature engineering performed on the dataset was coded (without using any libraries) by the authors. The following steps explain the feature engineering:

1. The top ten positively and negatively correlated features (obtained by calculating Pearson's Correlation Coefficient) were concatenated into one data frame.
2. The indices of Step 1's features were stored in a list.
3. The minimum, maximum, and average values of the features selected in Step 1 were computed.
4. New column called 'common_index' was added to every feature obtained from Step 3. The 'common_index' column contained the indices of these new features.
5. The new feature combinations obtained from Step 3 were added to the corresponding index of Step 1's data frame.
6. Step 5's data frame was added to the standardized and one-hot encoded training dataset.
7. The 'common_index' column was dropped from Step 6's data frame.

The code repository is on GitHub and can be accessed at this link.

6 Contributions

The following list shows the contributions of the authors:

1. Completion of exploratory data analysis- (Sudarshana).
2. Completion of data preprocessing- (Kranti).
3. Completion of the trivial system- (Sudarshana).
4. Completion of the baseline system- (Kranti).
5. Completion of feature engineering- (Kranti).
6. Completion of dimensionality adjustment- (Kranti & Sudarshana).
7. Completion of cross-validation- (Kranti & Sudarshana).
8. Completion of model implementation- (Kranti & Sudarshana).
9. Comparison of models and selection of the best model- (Sudarshana).
10. Completion of the project report- (Kranti & Sudarshana).

7 Summary and Conclusions

In this project, the authors analyzed the effect of feature engineering and dimensionality reduction techniques (performed on the dataset) on the performance of different models. The minimum, maximum, and mean were computed (as a feature engineering method) to help classify the mushrooms. This project carried out two experiments to observe the classification behaviors of the different models. The first experiment involved the usage of Principal Component Analysis on the feature-engineered dataset, and the second experiment involved the application of Univariate Feature Selection to the one-hot encoded and standardized dataset (without feature engineering). Seven models- Random Forest Classifier, Logistic Regression, Support Vector Machine, Multi-layer Perceptron, Naive Bayes Classifier, Trivial, and Baseline Systems were trained and tested on the data. It was observed that the models performed well on pre-processed and feature-engineered dataset compared to performance on only the pre-processed dataset. The future scope of this project would be to explore the effects of hyperparameter tuning on these models and to study the impact of Transfer Learning on the models.

References

[1] Dua, D. and Graff, C.(2019), "UCI Machine Learning Repository", Irvine, CA: University of California, School of Information and Computer Science.

[2] Sci-Kit-learn webpage- cross validation, "3.1. Cross-validation: evaluating estimator performance" (accessed April 17, 2023).

[3] Sci-Kit-learn webpage- standardization, "sklearn.preprocessing.StandardScaler" (accessed April 15, 2023).

[4] Sci-Kit-learn webpage- Pearson's Correlation Coefficient, "sklearn.feature_selection.r_regression" (accessed April 15, 2023).

[5] Sci-Kit-learn webpage- UFS, "sklearn.feature_selection.chi2" (accessed April 18, 2023).

[6] Sci-Kit-learn webpage- PCA, "sklearn.decomposition.PCA" (accessed April 20, 2023).

[7] Sci-Kit-learn webpage- NMC, "sklearn.neighbors.NearestCentroid" (accessed April 21, 2023).

[8] Sci-Kit-learn webpage- Logistic Regression, "sklearn.linear_model.LogisticRegression" (accessed April 21, 2023).

[9] Sci-Kit-learn webpage- SVM, "sklearn.svm.SVC" (accessed April 21, 2023).

[10] Khalid Salama, "Keras webpage- MLP" (accessed April 21, 2023).

[11] Sci-Kit-learn webpage- Random Forest Classifier, "sklearn.ensemble.RandomForestClassifier" (accessed April 23, 2023).

[12] Sci-Kit-learn webpage- NB, "sklearn.naive_bayes.GaussianNB" (accessed April 23, 2023).